

A Note on the Exact Schedulability Analysis for Segmented Self-Suspending Systems^{*}

Jian-Jia Chen

Department of Informatics, TU Dortmund University, Germany

Abstract. This report considers a sporadic real-time task system with n sporadic tasks on a uniprocessor platform, in which the lowest-priority task is a segmented self-suspension task and the other higher-priority tasks are ordinary sporadic real-time tasks. This report proves that the schedulability analysis for fixed-priority preemptive scheduling even with only one segmented self-suspending task as the lowest-priority task is coNP -hard in the strong sense. Under fixed-priority preemptive scheduling, Nelissen et al. in ECRTS 2015 provided a mixed-integer linear programming (MILP) formulation to calculate an upper bound on the worst-case response time of the lowest-priority self-suspending task. This report provides a comprehensive support to explain several hidden properties that were not provided in their paper. We also provide an input task set to explain why the resulting solution of their MILP formulation can be quite far from the exact worst-case response time.

1 Introduction and Models

We consider a system \mathbf{T} of n sporadic real-time tasks. A sporadic task τ_i in \mathbf{T} releases an infinite number of jobs that arrive with the minimum inter-arrival time constraint. A sporadic real-time task τ_i is characterized by its *worst-case execution time* C_i , its *minimum inter-arrival time* (also called period) T_i and its *relative deadline* D_i . In addition, each job of task τ_i has also a specified worst-case self-suspension time S_i . When a job of task τ_i arrives at time t , the job should finish no later than its *absolute deadline* $t + D_i$, and the next job of task τ_i can only be released no earlier than $t + T_i$. If the relative deadline D_i of task τ_i in the task set is always equal to (no more than, respectively) the period T_i , such a task set is called a *implicit-deadline* (*constrained-deadline*, respectively) task set (system). If $D_i > T_i$ for a certain task τ_i in \mathbf{T} , then the task system is an *arbitrary-deadline* task system. The response time of a job is defined as its finishing time minus its release (arrival) time. The worst-case response time $WCRT_i$ is the upper bound on the response times of all the jobs of task τ_i . The *response time analysis* of a task τ_i under a scheduling algorithm is to provide a safe upper bound on $WCRT_i$.

There are two typical models for self-suspending sporadic task systems: 1) the dynamic self-suspension task model, and 2) the segmented self-suspension task model. In the *dynamic* self-suspension task model, e.g., [1, 2, 6, 12, 13, 18, 20], in addition to the worst-case execution time C_i of sporadic task τ_i , we have also the worst-case self-suspension time S_i of task τ_i . In the *segmented* self-suspension task model, e.g., [4, 5, 10,

^{*} This report has been supported by DFG, as part of the Collaborative Research Center SFB876 (<http://sfb876.tu-dortmund.de/>).

[11, 14, 22], the execution behaviour of a job of task τ_i is specified by interleaved computation segments and self-suspension intervals. The dynamic self-suspension model provides a simple specification by ignoring the juncture of I/O access, computation offloading, or synchronization. However, if the suspending behaviour can be characterized by using a segmented pattern, the segmented self-suspension task model can be more appropriate.

This report considers a *segmented* self-suspension task model. If a task τ_i can suspend itself, a job of task τ_i is further characterized by the computation segments and suspension intervals as an array $(C_i^1, S_i^1, C_i^2, S_i^2, \dots, S_i^{m_i-1}, C_i^{m_i})$, composed of m_i computation segments separated by $m_i - 1$ suspension intervals.

In this report, we will *only* consider the following special case in fixed-priority (FP) preemptive scheduling:

- Task τ_n is the lowest-priority task and is a segmented self-suspension task. We will further assume that $D_n \leq T_n$.
- There are $n-1$ higher-priority tasks, $\tau_1, \tau_2, \dots, \tau_{n-1}$. These $n-1$ tasks are indexed from the highest priority τ_1 to the lowest priority τ_{n-1} . The task set $\{\tau_1, \tau_2, \dots, \tau_{n-1}\}$ can be an arbitrary-, constrained-, or implicit-deadline task set.

Since we only have one self-suspending task in this report, we use m to denote m_n for notational simplicity, where $m \geq 2$. Moreover, the arrival time of a computation segment is defined as the moment when the computation segment is ready to be executed, after all its previous computation segments and self-suspension intervals are done. In this report, the response time R_j of a computation segment C_n^j is defined as the finishing time of the computation segment minus the arrival time of the computation segment.

We consider uniprocessor fixed-priority preemptive scheduling. We say that a release pattern of the tasks in \mathbf{T} is *valid* if the jobs of the tasks in \mathbf{T} do not violate any of the temporal characteristics regarding to the minimum inter-arrival time, worst-case execution time, and worst-case self-suspension time. We say that a schedule is *feasible* if all the deadlines are met for a valid release pattern of the tasks in \mathbf{T} . Moreover, a task system (set) is *schedulable* by a scheduling algorithm if the resulting schedule for any valid release pattern of \mathbf{T} is always feasible. A *schedulability test* of a scheduling algorithm for a given task system is to validate whether the task system is schedulable by the scheduling algorithm. A *sufficient* schedulability test provides only sufficient conditions for validating the schedulability of a task system. A *necessary* schedulability test provides only necessary conditions to allow the schedulability of a task system. An *exact* schedulability test provides necessary and sufficient conditions for validating the schedulability.

By the assumption of fixed-priority preemptive scheduling, a schedulability test of task τ_i can be done by removing all the lower-priority tasks, $\tau_{i+1}, \dots, \tau_n$. Since task τ_n is the lowest-priority task, the schedulability test of the $n-1$ higher-priority tasks under the given FP preemptive scheduling can be done by using the well-known response time analysis, [16, 17].

Therefore, the remaining problem is to validate whether the segmented self-suspension task τ_n is schedulable by FP preemptive scheduling (as the lowest-priority task). For this problem, the only results in the literature were provided by Lakshmanan and Rajkumar [15] and Nelissen et al. [21]. Lakshmanan and Rajkumar proposed a pseudo-

polynomial-time worst-case response time analysis, by revising the well-known critical instant theorem originally defined in [19]. This has been recently disproved by Nelissen et al. [21]. The schedulability test by Nelissen et al. [21] requires exponential-time complexity even for such a case when the task system has *only one self-suspending task*. Furthermore, Nelissen et al. [21] also assumed that all the tasks are with constrained deadlines, i.e., $D_i \leq T_i$ for every task $\tau_i \in \mathbf{T}$. The other solutions [10, 22] require pseudo-polynomial time complexity but are only sufficient schedulability tests.

Regarding to computational complexity, it was shown by Ridouard et al. [23] that the scheduler design problem for the segmented self-suspension task model is \mathcal{NP} -hard in the strong sense.¹ The proof in [23] only needs each segmented self-suspending task to have one self-suspension interval with two computation segments. It was reported by Chen et al. [7] that the schedulability test problem in several cases is also strongly coNP -hard under dynamic-priority scheduling, in which the priority of a job may change over time. Such observations made in [7] are based on the special cases to reduce from the schedulability test problem of the ordinary constrained-deadline sporadic task systems (without self-suspension), which has been recently proved to be coNP -hard in the strong sense by Ekberg and Wang [8] under earliest-deadline-first (EDF) scheduling. For fixed-priority (FP) preemptive scheduling, in which a task is assigned a fixed priority level, the computational complexity of the schedulability test problem for the segmented self-suspension task model is open.

Contribution: This report provides the following results of the schedulability test problem and the worst-case response time analysis for self-suspending sporadic task systems:

- In Section 3, we prove that the schedulability analysis for fixed-priority (FP) preemptive scheduling even with only one segmented self-suspending task as the lowest-priority task is coNP -hard in the strong sense when there are more than one self-suspension interval (or equivalently more than two computation segments). The computational complexity analysis is valid for both implicit-deadline and constrained-deadline cases, when the priority assignment is given. Our proof also shows that validating whether there exists a feasible priority assignment is also coNP -hard in the strong sense for constrained-deadline segmented self-suspending task systems.
- This report shows that the upper bound on the worst-case response time derived from the MILP developed by Nelissen et al. [21] can be very far from the actual worst-case response time in Section 5.

2 Proof for the Necessary Condition of Worst-Case Response Time

Let σ be a fixed-priority preemptive schedule for a valid release pattern RP of the task system \mathbf{T} . We consider two cases:

- Case 1 when all the jobs of task τ_n in schedule σ have their response times no more than T_n : We pick an arbitrary job J of task τ_n from σ . For this case, removing all the other jobs of task τ_n (except J) from σ does not change the schedule of the remaining jobs in σ .

¹ Ridouard et al. [23] termed this problem as the feasibility problem for the decision version to verify the existence of a feasible schedule.

- Case 2 when there exists a job of task τ_n in schedule σ , in which the response time of the job is larger than T_n . Let J be the first job in the schedule σ in which the response time of J is strictly larger than T_n . For this case, removing all the other jobs of task τ_n (except J) from σ does not change the schedule of the remaining jobs in σ .

In both cases, for $j = 1, 2, \dots, m$, suppose that the arrival time and finishing time of the j -th computation segment of job J is g_j and f_j , respectively. In both cases, by definition, $g_1 \leq f_1 \leq g_2 \leq f_2 \leq \dots \leq g_m \leq f_m$, and $f_m - g_1 \leq WCRT_n$.

The following lemmas provide the necessary conditions for the worst-case release patterns for both cases. Specifically, Condition 1 in Lemma 1 was also provided in Lemma 2 by Nelissen et al. in [21]. For the completeness of this report and the correctness of the MILP, we also include the proof of Condition 1 in Lemma 1 here.

Lemma 1. *If $WCRT_n \leq T_n$, then the worst-case response time of task τ_n happens (as necessary conditions) when*

- Condition 1: all the higher-priority tasks $\tau_1, \tau_2, \dots, \tau_{n-1}$ only release their jobs in time intervals $[g_j, f_j)$ for $j = 1, 2, \dots, m$, and*
Condition 2: $g_{j+1} - f_j$ is always S_n^j , $\forall j = 1, 2, \dots, m - 1$, and
Condition 3: all the jobs are executed with their worst-case execution times.

Proof. We prove this lemma by showing that job J defined at the opening of this section can only increase its response time by following these three conditions. By the assumption $WCRT_n \leq T_n$, we consider Case 1.

We start with Condition 1. Suppose that the schedule σ has to execute certain higher-priority jobs in time interval $[t_1, g_1)$ and the processor idles right before t_1 . That is, the processor does not idle between t_1 and g_1 . In this case, we can change the arrival time of the computation segment C_n^1 of job J from g_1 to t_1 . This change in the release pattern RP does not change the resulting preemptive schedule σ , but the response time of J becomes $f_m - t_1 \geq f_m - g_1$ since $t_1 \leq g_1$.

For $j = 2, 3, \dots, m$, suppose that the schedule σ has to execute certain higher-priority jobs to keep the processor busy in time interval $[t_j, g_j)$ and the processor either idles or executes job J right before t_1 . By definition, for $j = 2, 3, \dots, m$, we know that $t_j \geq f_{j-1}$; otherwise the $(j-1)$ -th computation segment of job J cannot be finished at time f_{j-1} . Similarly, we can change the arrival time of the computation segment C_n^j of job J from g_j to t_j . This change in the release pattern RP does not change the resulting preemptive schedule σ nor the response time of J .

Let g_j for $j = 1, 2, \dots, m$ be the revised arrival time of the computation segment C_n^j of job J changed above. After we change the release pattern of job J , we know that the suspension time between the two computation segments C_n^j and C_n^{j+1} of job J is exactly $g_{j+1} - f_j \leq S_n^j$ for $j = 1, 2, \dots, m - 1$. Therefore, the revised release pattern remains valid.

Now, we can safely remove the higher jobs released before g_1 , after or at f_m , and in time intervals $[f_1, g_2), [f_2, g_3), \dots, [f_{m-1}, g_m)$. Since these jobs do not (directly or indirectly) interfere in the execution of job J at all, removing them does not have any impact on the execution of job J . Again, let RP be the revised release pattern, and let σ be its corresponding FP preemptive schedule. Now, Condition 1 holds.

We now revise the release pattern of the higher-priority jobs and J for Condition 2. We start from $j = 2$. If $g_j - f_{j-1} < S_n^{j-1}$, then we greedily perform the following steps:

- First, any higher-priority jobs released after or at g_j are delayed exactly by $S_n^{j-1} - (g_j - f_{j-1})$ time units.
- Second, the $(j - 1)$ -th suspension interval of job J is increased to suspend for exactly S_n^{j-1} time units, and g_ℓ is set to $g_\ell + S_n^{j-1} - (g_j - f_{j-1})$ for $\ell = j, j + 1, \dots, m$.

With the above two steps, the schedule σ remains almost unchanged by just adding $S_n^{j-1} - (g_j - f_{j-1})$ amount of idle time. We repeat the above procedure for $j = 2, 3, \dots, m$. Again, let σ be the above revised schedule with the revised release pattern. Now, after the adjustment, Condition 2 holds, and the response time of job J in this schedule is larger than or equal to the original one.

Condition 3 is rather trivial. If a job in schedule σ has a shorter execution time, we can increase its execution time to its worst-case execution time. We can then adjust the release pattern with a similar procedure like the operations for Condition 2 to increase the response time of job J .

With the above discussions, we reach the conclusion of this lemma. \square

Lemma 2. *If $WCRT_n > T_n$, the response time of task τ_n in a release pattern that satisfies Conditions 1, 2, and 3 in Lemma 1 is larger than T_n .*

Proof. By the assumption $WCRT_n > T_n$, we consider that there exists a schedule σ in which Case 2 (at the opening of the section) holds. The rest of the proof is identical to the proof of Lemma 1. \square

We now demonstrate a few properties based on Lemma 1. By Lemma 1, for obtaining the (exact) worst-case response time of task τ_n , we simply have to examine all the release patterns that satisfy the three conditions in Lemma 1. Specifically, Condition 1 of Lemma 1 implies that we can set an *offset* variable $O_{i,j}$ (with $O_{i,j} \geq 0$) to define the release time of the first job of task τ_i , arrived no earlier than g_j . That is, for a given $O_{i,j}$, the first job released by task τ_i no earlier than g_j is released at time $O_{i,j} + g_j$.

With the above definitions, we can have the following properties. These properties can be used to reduce the search space for the worst-case response time of task τ_n . The first property was also provided in Corollary 1 in the paper by Nelissen et al. [21].

Property 1. For a higher-priority task τ_i , there must be at least one $O_{i,j}$ equal to 0.

Proof. This is quite trivial. By Lemma 1, either task τ_i does not release any job to interfere in any computation segment of job J or τ_i releases at least one job to interfere in certain computation segments of job J . For the former case, we can set $O_{i,1}$ to 0, which does not decrease the resulting worst-case response time. For the latter case, if all $O_{i,j} > 0$ for $j = 1, 2, \dots, m$, let j^* be the earliest computation segment of job J where task τ_i releases some jobs to interfere in. We can greedily set O_{i,j^*} to 0, which does not reduce the resulting worst-case response time. \square

Property 2. If the period of task τ_i is small enough, then the following property holds:

- **Case when $j = 1$:** If $T_i \leq S_n^1$, then $O_{i,1}$ is 0.

- **Case when $j = m$:** If $T_i \leq S_n^{m-1}$, then $O_{i,m}$ is 0.
- **Case when $2 \leq j \leq m - 1$:** If $T_i \leq S_n^{j-1}$ and $T_i \leq S_n^j$ then $O_{i,j}$ is 0.

That is, task τ_i releases one job together with the j -th computation segment of the job J . Moreover, task τ_i also releases the subsequent jobs strictly periodically with period T_i until the j -th computation segment of job J finishes.

Proof. The first case is clear due to Condition 2 in Lemma 1, i.e., the suspension time of the first suspension interval is exactly S_n^1 , since the release pattern of task τ_i to interfere in the first computation segment of job J is independent from the other computation segments. The second case is similar.

For any j with $2 \leq j \leq m - 1$, the condition $T_i \leq S_n^{j-1}$ implies that the release pattern of task τ_i to interfere in the $(j - 1)$ -th computation segment of job J is independent from the release pattern to interfere in the j -th computation segment. Similarly, the condition $T_i \leq S_n^j$ implies that the release pattern of task τ_i to interfere in the j -th computation segment of job J is independent from the release pattern of task τ_i to interfere in the $(j + 1)$ -th computation segment. Therefore, when $T_i \leq S_n^{j-1}$ and $T_i \leq S_n^j$, the release pattern of task τ_i to interfere in the j -th computation segment of job J is independent from the other computation segments.

Moreover, when the release pattern of task τ_i to interfere in the j -th computation segment of job J is independent from the other computation segments, the worst-case release pattern of task τ_i to interfere in the j -th computation segment of job J is to release 1) one job together with the j -th computation segment of the job J , and 2) the subsequent jobs strictly periodically with period T_i until the j -th computation segment of job J finishes. \square

Property 3. If $T_i \geq T_n$ and $WCRT_n \leq T_n$, then a higher-priority task τ_i only releases one job together with one of the m computation segments of the job (under analysis) of task τ_n .

Proof. This comes from Condition 1 in Lemma 1 and Property 1. \square

Property 4. If $T_i - C_i$ is small enough, then the following property of task τ_i holds:

- **Case when $j = 1$:** If $T_i - C_i \leq S_n^1$, then $O_{i,1}$ is 0.
- **Case when $j = m$:** If $T_i - C_i \leq S_n^{m-1}$, then $O_{i,m}$ is 0.
- **Case when $2 \leq j \leq m - 1$:** If $T_i - C_i \leq S_n^{j-1}$ and $T_i - C_i \leq S_n^j$ then $O_{i,j}$ is 0.

That is, task τ_i releases one job together with the j -th computation segment of the job J . Moreover, task τ_i also releases the subsequent jobs strictly periodically with period T_i until the j -th computation segment of job J finishes.

Proof. This is a simple extension of Property 2. \square

3 Computational Complexity

In this section, we will prove that the schedulability test problem for FP preemptive scheduling even with only one *segmented* self-suspending task as the lowest-priority task in the task system is $co\mathcal{NP}$ -hard in the strong sense. Specifically, we will also

show that our reduction implies that finding whether there exists a feasible priority assignment under FP scheduling for constrained-deadline task systems is also $\text{co}\mathcal{NP}$ -hard in the strong sense. We will first consider constrained-deadline task systems and then revise the reduction to consider implicit-deadline task systems.

Our reduction is from the 3-PARTITION problem [9]:²

Definition 1 (3-PARTITION Problem). *We are given a positive integer V , a positive integer M , and a set of $3M$ integer numbers $\{v_2, v_3, \dots, v_{3M+1}\}$ with the condition $\sum_{i=2}^{3M+1} v_i = MV$, in which $1 \leq V/4 < v_i < V/2$ and $M \geq 3$. Therefore, $V \geq 3$.*

Objective: *The problem is to partition the given $3M$ integer numbers into M disjoint sets $\mathbf{V}_1, \mathbf{V}_2, \dots, \mathbf{V}_M$ such that the sum of the numbers in each set \mathbf{V}_i for $i = 1, 2, \dots, M$ is V , i.e., $\sum_{v_j \in \mathbf{V}_i} v_j = V$. \square*

The decision version of the 3-PARTITION problem to verify whether such a partition into M disjoint sets exists or not is known \mathcal{NP} -complete in the strong sense [9] when $M \geq 3$.

3.1 Constrained-Deadline Task Systems

Definition 2 (Reduction to a constrained-deadline system). *For a given input instance of the 3-PARTITION problem, we construct $n = 3MV + 2$ sporadic tasks as follows:*

- For task τ_1 , we set $C_1 = V, S_1 = 0, D_1 = V, T_1 = 3V$.
- For task τ_i with $i = 2, 3, \dots, 3M + 1$, we set $C_i = v_i, S_i = 0, T_i = 21MV$ and $D_i = 3MV/2$ if M is an even number or $D_i = 3MV/2 + V/2$ if M is an odd number.
- For task τ_{3M+2} , we create a segmented self-suspending task with M computation segments separated by $M-1$ self-suspension intervals³, i.e., $m = m_{3M+2} = M$, in which $C_{3M+2}^j = V+1$ for $j = 1, 2, \dots, M$, $S_{3M+2}^j = 6V$ for $j = 1, 2, \dots, M-1$, $D_{3M+2} = M(4V+1) - V + 6V(M-1) = 10MV + M - 7V$, and $T_{3M+2} = 21MV$.

Due to the stringent relative deadline of task τ_1 , it must be assigned as the highest-priority task. Moreover, the $3M$ tasks, i.e., $\tau_2, \tau_3, \dots, \tau_{3M+1}$, created by using the integer numbers from the 3-PARTITION problem instance are assigned lower priorities than task τ_1 and higher priorities than task τ_{3M+2} . Since the integer numbers in the 3-PARTITION problem instance are given in an arbitrary order, without loss of generality, we index the tasks in $\tau_2, \tau_3, \dots, \tau_{3M+1}$ by the given priority assignment, i.e., a lower-indexed task has higher priority. (In fact, we can also assign all these $3M$ tasks with the same priority level.) \square

For the rest of the proof, the task set created in Definition 2 is referred to as \mathbf{T}^{red} .

Lemma 3. *Tasks $\tau_1, \tau_2, \dots, \tau_{3M+1}$ in \mathbf{T}^{red} can meet their deadlines under the specified FP scheduling.*

² For notational consistency and brevity in our reduction, we index the $3M$ integer numbers from 2.

³ The first version of the proof uses $6V$ for S_{3M+2}^j for $j = 1, 2, \dots, M-1$. By applying Property 3, we can also set S_{3M+2}^j to $2V$ and $D_{3M+2} = M(4V+1) - V + 2V(M-1) = 6MV + M - 3V$.

Proof. In FP scheduling, the segmented self-suspending task τ_{3M+2} in \mathbf{T}^{red} has no impact on the schedule of the higher-priority tasks. Therefore, we can use the standard schedulability test for FP scheduling to verify their schedulability. The schedulability of task τ_1 is obvious since $C_1 \leq D_1$. For $i = 2, 3, \dots, 3M+1$, task τ_i is schedulable under FP scheduling since $C_i + \sum_{j=1}^{i-1} \left\lceil \frac{D_i}{T_j} \right\rceil C_j = \left\lceil \frac{D_i}{T_1} \right\rceil C_1 + \sum_{j=2}^i C_j \leq \left\lceil \frac{D_i}{3V} \right\rceil V + MV = D_i$, where the last equality is due to $\left\lceil \frac{D_i}{3V} \right\rceil = \left\lceil \frac{3MV/2}{3V} \right\rceil = M/2$ when M is an even number and $\left\lceil \frac{D_i}{3V} \right\rceil = \left\lceil \frac{3MV/2+V/2}{3V} \right\rceil = (M+1)/2$ when M is an add number. \square

The worst-case response time of task τ_{3M+2} happens by using one of the release patterns with the conditions in Lemma 4:

Lemma 4. *The worst-case response time of task τ_{3M+2} in \mathbf{T}^{red} under FP scheduling happens under the following necessary conditions:*

1. Task τ_{3M+2} releases a job at time 0. This job requests the worst-case execution time per computation segment and suspends in each self-suspension interval exactly equal to its worst case.
2. Task τ_1 always releases one job together with each computation segment of the job (released at time 0) of task τ_{3M+2} , and releases the subsequent jobs strictly periodically with period $3V$ until a computation segment of task τ_{3M+2} finishes. Task τ_1 never releases any job when task τ_{3M+2} suspends itself.
3. For $i = 2, 3, \dots, 3M+1$, task τ_i only releases one job together with one of the M computation segments of the job (released at time 0) of task τ_{3M+2} .

All the jobs and all the computation segments are executed with their worst-case execution time specifications.

Proof. For task τ_1 in \mathbf{T}^{red} , since $T_1 < S_n^j$ for any $j = 1, 2, \dots, M-1$, the release pattern of task τ_1 is independent from the computation segments. This is formally proved in Property 2 in Section 2. Moreover, since $T_i > D_n$ for $i = 2, 3, \dots, 3M+1$, such a higher-priority task τ_i in \mathbf{T}^{red} only releases one job together with one of the M computation segments of the job (under analysis) of task τ_n . This is formally proved in Property 3 in Section 2. By putting all the above conditions together, we reach the conclusion for Lemma 4. \square

For the j -th computation segment of task τ_{3M+2} , suppose that $\mathbf{T}_j \subseteq \{\tau_2, \tau_3, \dots, \tau_{3M+1}\}$ is the set of the tasks released together with C_{3M+2}^j (under the third condition in Lemma 4). For notational brevity, let w_j be $\sum_{\tau_i \in \mathbf{T}_j} C_i$. By definition, w_j is a non-negative integer. Together with the second condition in Lemma 4, we can use the standard time demand analysis to analyze the worst-case response time R_j of the j -th computation segment of task τ_{3M+2} (after it is released) under the higher-priority interference due to $\{\tau_1\} \cup \mathbf{T}_j$. The response time R_j of a computation segment C_{3M+2}^j is defined as the finishing time of the computation segment minus the arrival time of the computation segment.

For a given task set \mathbf{T}_j (i.e., a given non-negative integer w_j), R_j is the minimum t with $t > 0$ such that

$$C_{3M+2}^j + \left(\sum_{\tau_i \in \mathbf{T}_j} C_i \right) + \left\lceil \frac{t}{T_1} \right\rceil C_1 = V + 1 + w_j + \left\lceil \frac{t}{3V} \right\rceil V = t.$$

Since R_j only depends on the non-negative integer w_j , we use $R(w_j)$ to represent R_j for a given \mathbf{T}_j . We know that $V + 1 + w_j + \lceil \frac{t}{3V} \rceil V = t$ happens with $\ell \cdot 3V < t \leq (\ell + 1) \cdot 3V$ for a certain non-negative integer ℓ . That is, $V + 1 + w_j + \lceil \frac{t}{3V} \rceil V > t$ when t is $\ell \cdot 3V$ and $V + 1 + w_j + \lceil \frac{t}{3V} \rceil V \leq t$ when t is $(\ell + 1)3V$. We know that ℓ is $\lceil \frac{V+1+w_j}{2V} \rceil - 1$. Moreover,

$$\begin{aligned} R(w_j) &= \ell \cdot 3V + V + (V + 1 + w_j - \ell \cdot 2V) \\ &= 2V + 1 + w_j + \ell \cdot V \\ &= V + 1 + w_j + \left\lceil \frac{V+1+w_j}{2V} \right\rceil V. \end{aligned}$$

This leads to three cases that are of interest:

$$R(w_j) = \begin{cases} 2V + 1 + w_j & \text{if } w_j \leq V - 1 \\ 4V + 1 & \text{if } w_j = V \\ V + 1 + w_j + \left\lceil \frac{V+1+w_j}{2V} \right\rceil V & \text{if } w_j > V \end{cases} \quad (1)$$

For example, if $w_j = 3V - 1$, then $R(w_j)$ is $6V$; if w_j is $3V$, then $R(w_j)$ is $7V + 1$.

With the above discussions, we can now conclude that the unique condition when task τ_{3M+2} misses its deadline in the following lemma.

Lemma 5. Suppose that $\mathbf{T}_j \subseteq \{\tau_2, \tau_3, \dots, \tau_{3M+1}\}$ and $\mathbf{T}_i \cap \mathbf{T}_j = \emptyset$ when $i \neq j$. Let $w_j = \sum_{\tau_i \in \mathbf{T}_j} C_i$. If a task partition $\mathbf{T}_1, \mathbf{T}_2, \dots, \mathbf{T}_M$ exists such that $\sum_{j=1}^M R(w_j) > M(4V + 1) - V$ with $R(w_j)$ defined in Eq. (1), then task τ_{3M+2} misses its deadline in the worst case; otherwise, task τ_{3M+2} always meets its deadline.

Proof. By Lemma 4, task τ_{3M+2} in \mathbf{T}^{red} is not schedulable under the fixed-priority preemptive scheduling if and only if there exists a task partition $\mathbf{T}_1, \mathbf{T}_2, \dots, \mathbf{T}_M$ such that $\sum_{j=1}^M R(w_j) + \sum_{j=1}^{M-1} S_{3M+2}^j = 6(M - 1)V + \sum_{j=1}^M R(w_j) > D_{3M+2} = M(4V + 1) + 6V(M - 1) - V$. This concludes the proof. \square

Instead of investigating the combinations of the task partitions, we analyze the corresponding total worst-case response time $\sum_{j=1}^M R(w_j)$ for the M computation segments of task τ_{3M+2} (by excluding the self-suspension time) by considering different non-negative integer assignments w_1, w_2, \dots, w_M with $\sum_{i=1}^M w_i = MV$ and $w_i \geq 0$ in the following lemmas.

Lemma 6. If $w_1 = w_2 = \dots = w_M = V$, then

$$\sum_{j=1}^M R(w_j) = M(4V + 1),$$

where $R(w_j)$ is defined in Eq. (1).

Proof. This comes directly by Eq. (1). \square

Lemma 7. For any non-negative integer assignment for w_1, w_2, \dots, w_M with $\sum_{i=1}^M w_i = MV$, if there exists a certain index j with $w_j \neq V$, then

$$\sum_{j=1}^M R(w_j) \leq M(4V + 1) - V,$$

where $R(w_j)$ is defined in Eq. (1).

Proof. Let \mathbf{X} be the set of indexes such that $0 \leq w_j < V$ for any $j \in \mathbf{X}$. Similarly, let \mathbf{Y} be the set of indexes such that $V < w_j$ for any $j \in \mathbf{Y}$. If $j \notin \mathbf{X} \cup \mathbf{Y}$, then $w_j = V$.

If there exists j in \mathbf{Y} with $w_j > 2V$, since $\sum_{i=1}^M w_i = MV$, there must exist an index i in \mathbf{X} with $w_i < V$. We can increase w_i to $w'_i = V$, which increases the worst-case response time $R(w_i)$ by $2V - w_i$ (i.e., from $2V + 1 + w_i$ to $4V + 1$). Simultaneously, we reduce w_j to $w'_j = w_j - (V - w_i) > V$. Therefore, $w_i + w_j = w'_i + w'_j$. Moreover, the reduction of w_j to w'_j also reduces the worst-case response time $R(w_j)$ by case 1) $V - w_i$ if $\left\lceil \frac{V+1+w'_j}{2V} \right\rceil$ is equal to $\left\lceil \frac{V+1+w_j}{2V} \right\rceil$, and by case 2) $V - w_i + V$ if $\left\lceil \frac{V+1+w'_j}{2V} \right\rceil$ is not equal to $\left\lceil \frac{V+1+w_j}{2V} \right\rceil$. In both cases, we can easily see that the worst-case response time is not decreased in the new integer assignment. Moreover, the index j remains in \mathbf{Y} and the index i is removed from set \mathbf{X} . We repeat the above step until all the indexes j in \mathbf{Y} are with $w_j \leq 2V$.

It is clear that \mathbf{X} and \mathbf{Y} are both non-empty after the above step. For the rest of the proof, let \mathbf{X} and \mathbf{Y} be defined after finishing the above step. Therefore, the condition $w_j \leq 2V$ holds for any $j \in \mathbf{Y}$. Due to the pigeon-hole principle, when \mathbf{Y} is not an empty set, \mathbf{X} is also not an empty set. Moreover, for an element i in \mathbf{X} , there must be a subset $\mathbf{Y}' \subseteq \mathbf{Y}$ and an index $\ell \in \mathbf{Y}'$ such that

$$\sum_{j \in \mathbf{Y}'} (w_j - V) \geq V - w_i > \sum_{j \in \mathbf{Y}' \setminus \{\ell\}} (w_j - V).$$

That is, we want to adjust w_i to V (i.e., w_i is increased by $V - w_i$), and the set $\mathbf{Y}' \setminus \{\ell\}$ is not enough to match the integer adjustment $V - w_i$ and the set \mathbf{Y}' is enough to match the integer adjustment $V - w_i$. We now increase w_i to V , which increases the worst-case response time $R(w_i)$ by $2V - w_i$. Simultaneously, we reduce w_j to V for every $j \in \mathbf{Y}' \setminus \{\ell\}$ and reduce w_ℓ to $w'_\ell = w_\ell - (V - w_i - \sum_{j \in \mathbf{Y}' \setminus \{\ell\}} (w_j - V))$. Since $V < w_j \leq 2V$ for any $j \in \mathbf{Y}'$ before the adjustment, the adjustment reduces the worst-case response time $R(w_j)$ by $w_j - V$ if $j \neq \ell$ and reduces $R(w_\ell)$ by $w_\ell - w'_\ell$. Therefore, the adjustment reduces $\sum_{j \in \mathbf{Y}'} R(w_j)$ by exactly $V - w_i$. Therefore, the adjustment in this step to change w_i in \mathbf{X} and w_j in \mathbf{Y}' increases the overall worst-case response time by exactly V time units.

By adjusting with the above procedure repeatedly, we will reach the integer assignment $w_1 = w_2 = \dots = w_M = V$ with bounded increase of the worst-case response time. As a result, we can conclude that $\sum_{j=1}^M R(w_j) \leq M(4V + 1) - |\mathbf{X}|V$. By the assumption $\sum_{i=1}^M w_i = MV$ and the existence of $w_j \neq V$ for some j , we know that $|\mathbf{X}|$ must be at least 1. Therefore, we reach the conclusion. \square

We use an example to illustrate how the procedure in Lemma 7 operates. Suppose that $w_1 = 0, w_2 = 3.5V, w_3 = 0.4V, w_4 = 0.6V, w_5 = 1.5V, w_6 = 0$ when $M = 6$ and

w_1	$R(w_1)$	w_2	$R(w_2)$	w_3	$R(w_3)$	w_4	$R(w_4)$	w_5	$R(w_5)$	w_6	$R(w_6)$	\mathbf{X}	\mathbf{Y}	$\sum_{j=1}^6 R(w_j)$
0	$2V+1$	$3.5V$	$7.5V+1$	$0.4V$	$2.4V+1$	$0.6V$	$2.6V+1$	$1.5V$	$4.5V+1$	0	$2V+1$	$\{1, 3, 4, 6\}$	$\{2, 5\}$	$21V+6$
V	$4V+1$	$2.5V$	$5.5V+1$	—	—	—	—	—	—	—	—	$\{3, 4, 6\}$	$\{2, 5\}$	$21V+6$
—	—	$1.5V$	$4.5V+1$	—	—	—	—	—	—	V	$4V+1$	$\{3, 4\}$	$\{2, 5\}$	$22V+6$
—	—	V	$4V+1$	V	$4V+1$	—	—	$1.4V$	$4.4V+1$	—	—	$\{4\}$	$\{5\}$	$23V+6$
—	—	—	—	—	—	V	$4V+1$	$1V$	$4V+1$	—	—	\emptyset	\emptyset	$24V+6$

Table 1: An example of Lemma 7

V is an integer multiple of 10. We will start from $\mathbf{X} = \{1, 3, 4, 6\}$ and $\mathbf{Y} = \{2, 5\}$. As shown in Table 1, the operation makes $\sum_{j=1}^6 R(w_j)$ increase. Note that the conclusion $\sum_{j=1}^M R(w_j) \leq M(4V+1) - |\mathbf{X}|V$ in Lemma 7 was for $|\mathbf{X}| = \{3, 4\}$ in this example when $w_j < 2V$ for any $j \in \mathbf{Y}$.

We can now conclude the $\text{co}\mathcal{NP}$ -hardness.

Theorem 1. *The schedulability analysis for FP scheduling even with only one segmented self-suspending task as the lowest-priority task in the sporadic task system is $\text{co}\mathcal{NP}$ -hard in the strong sense, when the number of self-suspending intervals in the self-suspending task is larger than or equal to 2 and $D_i \leq T_i$ for every task τ_i .*

Proof. The reduction in Definition 2 requires polynomial time. Moreover, by Lemmas 4, 5, 6, and 7, a feasible solution of the 3-PARTITION problem for the input instance exists if and only if task τ_{3M+2} is not schedulable by the FP scheduling when $M \geq 3$. Therefore, this concludes the proof. \square

Corollary 1. *Validating whether there exists a feasible priority assignment is $\text{co}\mathcal{NP}$ -hard in the strong sense for constrained-deadline segmented self-suspending task systems.*

Proof. This comes directly from Theorem 1 and the only possible priority level for task τ_{3M+2} to be feasible in \mathbf{T}^{red} . \square

3.2 Implicit-Deadline Task Systems

The $\text{co}\mathcal{NP}$ -hardness in the strong sense for testing the schedulability of task τ_n under FP scheduling can be easily proved with the same input as in \mathbf{T}^{red} by changing the periods of the tasks as follows:

- For task τ_1 , we set $D_1 = 3V, T_1 = 3V$.
- For task τ_i with $i = 2, \dots, 3M+1$, we set $T_i = D_i = 10MV + M - 7V$.
- For task τ_{3M+2} , we set $T_{3M+2} = D_{3M+2} = 10MV + M - 7V$.

Assume that τ_{3M+2} is the lowest-priority task. It is not difficult to see that all the conditions in Lemma 4 still hold for testing whether task τ_{3M+2} can meet its deadline or not (but not for the worst-case response time if task τ_{3M+2} misses the deadline). Therefore, the schedulability analysis for FP scheduling even with only one segmented self-suspending task as the lowest-priority task in the sporadic task system is $\text{co}\mathcal{NP}$ -hard in the strong sense, when the number of self-suspending intervals in the self-suspending task is larger than or equal to 2 and $D_i = T_i$ for every task τ_i .

However, the above argument does not hold if we assign task τ_{3M+2} to the highest-priority level. Therefore, the above proof does not support a similar conclusion for implicit-deadline task systems to that for constrained-deadline task systems in Corollary 1.

4 MILP Approaches

Even though the properties in Lemma 1 provide the necessary conditions for the worst-case response time, finding the worst-case release pattern is in fact a hard problem as shown in the analysis in Section 3. However, if we can tolerate exponential time complexity, is there a strategy that can find the worst-case pattern based on Lemma 1 safely without performing exhaustive searches? One possibility is to model the problem as an MILP, which has been already presented by Nelissen et al. [21].

The worst-case response time analysis by Nelissen et al. [21] is based on the following mixed-integer linear programming (MILP):

$$\textbf{maximize:} \quad S_n + \sum_{j=1}^m R_j \quad (2a)$$

subject to:

$$R_j = C_n^j + \sum_{i=1}^{n-1} N_{i,j} \times C_i, \quad \forall j = 1, \dots, m \quad (2b)$$

$$O_{i,j} \geq 0, \quad \forall i = 1, \dots, n-1, \forall j = 1, \dots, m \quad (2c)$$

$$O_{i,j+1} \geq O_{i,j} + N_{i,j} \times T_i - (R_j + S_n^j), \quad \forall i = 1, \dots, n-1, \forall j = 1, \dots, m-1 \quad (2d)$$

$$0 \leq N_{i,j} \leq \left\lceil \frac{R_j - O_{i,j}}{T_i} \right\rceil, \quad \forall i = 1, \dots, n-1, \forall j = 1, \dots, m \quad (2e)$$

$$N_{i,j} \text{ is an integer}, \quad \forall i = 1, \dots, n-1, \forall j = 1, \dots, m \quad (2f)$$

$$R_j \leq UB_{ss,j} \quad \forall j = 1, 2, \dots, m, \quad (2g)$$

$$S_n + \sum_{j=1}^m R_j \leq UB_{ss} \quad (2h)$$

$$\text{Eq. (3) holds.} \quad (2i)$$

In the above MILP, the objective function $S_n + \sum_{j=1}^m R_j$ is the worst-case response time of task τ_n , where R_j is a variable (as a real number) that represents the response time of the j -th computation segment C_n^j of task τ_n . The variable $O_{i,j}$ defines the *offset* of the first job of a higher-priority task τ_i released no earlier than the arrival time of the j -th computation segment C_n^j of task τ_n . That is, if the arrival time of C_n^j is t_j , then the first job of task τ_i released at or after t_j is at time $t_j + O_{i,j}$. The integer variable $N_{i,j}$ defines the maximum number of jobs of a higher-priority task τ_i that are released to *successfully interfere* in the computation segment C_n^j of task τ_n .

The three additional constraints, expressed by Eq. (9), Eq. (11), and Eq. (16), in the MILP in [21] are expressed here by Eq. (2g), Eq. (2h), and Eq. (3), respectively. Here, UB_{ss} is defined as the upper bound on the worst-case response time of task τ_n , and $UB_{ss,j}$ is defined as the upper bound on the worst-case response time of the j -th computation segment of task τ_n . Later in this section, we will show that the condition in Eq. (2b) may over-estimate the worst-case response time. Therefore, the additional

constraint (expressed by Eq. (16), in the MILP in [21]) is used to reduce the pessimism as follows:

$$\forall i = 1, 2, \dots, n-1, j = 1, 2, \dots, m, \quad R_j > rel_{i,j} + \sum_{\ell=1}^{n-1} \max \left\{ 0, \left\lfloor \frac{O_{\ell,j} + N_{\ell,j}T_{\ell} - rel_{i,j}}{T_{\ell}} \right\rfloor C_{\ell} \right\}, \quad (3)$$

where $rel_{i,j} = O_{i,j} + (N_{i,j} - 1)T_i$. This means that the (total) execution time of all the higher-priority jobs (by tasks $\tau_1, \tau_2, \dots, \tau_{n-1}$) released after $rel_{i,j}$ should be less than $R_j - rel_{i,j}$.

Here, we first explain why the MILP by utilizing only the constraints from Eq. (2b) to Eq. (2f) is already a safe (but *not tight/exact*) result based on Lemma 1. Therefore, this also leads to the motivation to examine the pessimism by different combinations of the additional constraints Eq. (2g), Eq. (2h), and Eq. (3) in Section 5.

4.1 MILP by Using Lemma 1

We only consider the release patterns of the tasks in \mathbf{T} , where all the three conditions in Lemma 1 hold. Let $r_{i,j}$ be the arrival time of the first job of task τ_i arrived after or at time g_j in a concrete release pattern, in which all the three conditions in Lemma 1 hold. If task τ_i does not release any job after or at time g_j , we set $r_{i,j}$ to ∞ .⁴ By the minimum inter-arrival time constraint of task τ_i , we know that task τ_i cannot release any job in time interval $(r_{i,j+1} - T_i, r_{i,j+1})$. That is, in this release pattern, there are *at most* $\left\lfloor \frac{r_{i,j+1} - T_i - r_{i,j}}{T_i} \right\rfloor + 1 \leq \frac{r_{i,j+1} - r_{i,j}}{T_i}$ jobs from task τ_i that can interfere in the j -th computation segment of job J .

Let $N_{i,j}$ be the number of jobs of a higher-priority task τ_i released in time interval $[g_j, f_j]$ in this release pattern. By definition, $N_{i,j}$ is a non-negative integer. The maximum number of jobs that task τ_i can release in time interval $[r_{i,j}, f_j]$ in this release pattern can be expressed by the following inequality:

$$0 \leq N_{i,j} \leq \max \left\{ 0, \left\lfloor \frac{f_j - r_{i,j}}{T_i} \right\rfloor \right\}, \quad \forall i = 1, \dots, n-1, j = 1, \dots, m. \quad (4)$$

The reason to put $\max \left\{ 0, \left\lfloor \frac{f_j - r_{i,j}}{T_i} \right\rfloor \right\}$ instead of only $\left\lfloor \frac{f_j - r_{i,j}}{T_i} \right\rfloor$ in the right-hand side of Eq. (4) is to avoid the case that $\left\lfloor \frac{f_j - r_{i,j}}{T_i} \right\rfloor < 0$, which is possible if $r_{i,j} > f_j + T_i$.

There is one simple trick regarding to the setting of $r_{i,j}$. If $r_{i,j} > f_j + T_i$, for this release pattern, we know that 1) task τ_i does not release any job to interfere in the j -th computation segment of job J and 2) the number of jobs of task τ_i that are released to interfere in the $(j-1)$ -th computation segment of job J is purely dominated by $\max \left\{ 0, \left\lfloor \frac{f_{j-1} - r_{i,j-1}}{T_i} \right\rfloor \right\}$. Therefore, if $r_{i,j} > f_j + T_i$, we can safely set $r_{i,j}$ to $f_j + T_i$ (but we do not change the release pattern to release a job of task τ_i at time $f_j + T_i$ for such a case). With this, we can then rephrase Eq. (4) into

$$0 \leq N_{i,j} \leq \left\lfloor \frac{f_j - r_{i,j}}{T_i} \right\rfloor, \quad \forall i = 1, \dots, n-1, j = 1, \dots, m. \quad (5)$$

⁴ With the discussions below, we will later set $r_{i,j}$ to $f_j + T_j$ for such a case (but not release any job of task τ_i at time $f_j + T_j$).

By earlier discussions, we also have

$$N_{i,j} \leq \frac{r_{i,j+1} - r_{i,j}}{T_i}, \quad \forall i = 1, \dots, n-1, j = 1, \dots, m-1. \quad (6)$$

By Condition 1 and Condition 3 in Lemma 1, we also know that

$$f_j \leq g_j + C_n^j + \sum_{i=1}^{n-1} N_{i,j} \times C_i \quad \forall j = 1, 2, \dots, m. \quad (7)$$

Without loss of generality, we can set g_1 to 0. By Condition 2 in Lemma 1, we have

$$g_1 = 0 \text{ and } g_j = f_{j-1} + S_n^{j-1} \quad \forall j = 2, 3, \dots, m. \quad (8)$$

Now we can conclude the following theorem:

Theorem 2. Suppose that $g_j, f_j, r_{i,j}$ are variables of real numbers and $N_{i,j}$ are variables for non-negative integer numbers for $i = 1, 2, \dots, n-1$ and for $j = 1, 2, \dots, m$. The optimal solution of the following MILP is a safe upper bound on the worst-case response time of task τ_n if $WCRT_n \leq T_n$.

$$\textbf{maximize:} \quad f_m \quad (9a)$$

subject to:

$$r_{i,j} \geq g_j, \forall i = 1, \dots, n-1, \forall j = 1, \dots, m \quad (9b)$$

$$N_{i,j} \text{ is an integer}, \forall i = 1, \dots, n-1, \forall j = 1, \dots, m \quad (9c)$$

and Conditions in Eqs. (5), (6), (7), (8) hold.

Proof. This comes from the above discussions and Lemma 1. The release pattern that has the maximum f_m (provided that g_1 is set to 0) by using FP preemptive scheduling under all the constraints due to the three conditions in Lemma 1 leads to the worst-case response time if $WCRT_n \leq T_n$. \square

However, the MILP in Eq. (9) is not an exact response time analysis (or schedulability test) due to the following reason: the condition in Eq. (7) is only a safe upper bound on f_j , but does not provide the exact f_j under the release pattern. Suppose that n is 2. We have $T_1 = 4$ and $C_1 = 2$. Consider $g_1 = 0$ and $r_{1,1} = 0$, $C_n^1 = 2$, and $S_n^1 = 8$. In this case, it implies that the suspension interval S_n^1 has no impact when we analyze the worst-case finishing time of the first computation segment.⁵ It is clear that the exact (worst-case) finishing time of C_n^1 is 4 under this release pattern. However, there is another feasible solution that satisfies Eq. (7) by setting $N_{1,1}$ to 2, f_1 to 6, and $r_{1,2}$ to 14. Therefore, in fact, f_1 can have the following cases:⁶

- f_1 is 2 when $N_{1,1}$ is 0,
- f_1 is 4 when $N_{1,1}$ is 1, and
- f_1 is 6 when $N_{1,1}$ is 2.

However, due to the objective function for *maximization*, the optimal MILP solution is to set f_1 to 6 instead of 4 under this MILP.⁷

⁵ This is also proved in Property 2.

⁶ For this case, it becomes infeasible when $N_{1,1}$ is larger than 2.

⁷ This also explains why the statement in the earlier version of this report (<https://arxiv.org/abs/1605.00124v1>) was erroneous since it skipped the above discussion and directly concluded that the MILP returns the exact worst-case response time.

4.2 Connection to the MILP by Nelissen et al. in ECRTS 2015

The MILP in Eq. (9) looks different from the MILP in Eq. (2), but they are in fact equivalent. Suppose that $R_j = f_j - g_j, \forall j = 1, 2, \dots, m$ and $O_{i,j} = r_{i,j} - g_j, \forall i = 1, 2, \dots, n-1, \forall j = 1, 2, \dots, m$. We can rephrase the MILP in Eq. (9) into the MILP in Eq. (2) as follows:

- Clearly, the objective function in Eq. (9a) is identical to that in Eq. (2a).
- The condition in Eq. (7) leads to Eq. (2b).
- The condition in Eq. (9b) is identical to Eq. (2c).
- The condition in Eq. (8) and Eq. (7) can be used to rephrase Eq. (6) into

$$N_{i,j} \leq \frac{r_{i,j+1} - r_{i,j}}{T_i} = \frac{g_j + R_j + S_n^j + O_{i,j+1} - (g_j + O_{i,j})}{T_i} = \frac{R_j + S_n^j + O_{i,j+1} - O_{i,j}}{T_i},$$

which is identical to the condition in Eq. (2d).

- Moreover, the condition in Eq. (5) is identical to Eq. (2e).

Therefore, we have the following corollaries.

Corollary 2. *The optimal solution of the MILP in Eq. (2) (even by excluding Eqs. (2g), (2h), or (3)) is a safe upper bound of the worst-case response time of task τ_n if $WCRT_n \leq T_n$.*

Corollary 3. *If the optimal solution of the MILP in Eq. (2) (even by excluding Eqs. (2g), (2h), or (3)), or equivalently the MILP in Eq. (9) is no more than T_n , then $WCRT_n \leq T_n$.*

5 Response Time Analysis: How Far is the Gap?

Since the MILP approach listed in Section 4 does not provide the exact worst-case response time of task τ_n , it is also meaningful to examine whether the upper bound on the worst-case response time by using the MILP approach in Section 4 is always very close to (or not too far from) the exact worst-case response time. Unfortunately, we will demonstrate a task set, in which the derived worst-case response time from the MILP in Eq. (2) is at least $\frac{4m+4}{9}$ times the exact worst-case response time, where $m \geq 2$ is the number of computation segments of task τ_n .

We consider the following task set \mathbf{T}^{MILP} with $n = m + 4$ tasks, where q is a positive integer, m is a positive integer with $m \geq 2$, and $0 < \epsilon < 1/q$:

- For task τ_1 , we set $C_1 = 1, S_1 = 0, D_1 = T_1 = 2$.
- For task τ_2 , we set $C_2 = q, S_2 = 0, D_2 = T_2 = 4q$.
- For task τ_3 , we set $C_3 = 2q - 1 + \epsilon, S_3 = 0, D_3 = T_3 = 8q$.
- For task τ_i with $i = 4, 5, \dots, m+3$, we set $C_i = 1 - \epsilon, S_i = 0, D_i = 8qm, T_i = 16qm^2 + (m-1)(2q-1)$.
- For task τ_{m+4} , we create a segmented self-suspending task with m computation segments separated by $m-1$ self-suspension intervals, i.e., $m_n = m$, in which $C_{m+4}^j = 1 - \epsilon$ for $j = 1, 2, \dots, m$, $S_{m+4}^j = 2q - 1$ for $j = 1, 2, \dots, m-1$. The values of D_{m+4} and T_{m+4} are left open, and our goal here is to find the minimum feasible D_{m+4} that can be set when T_{m+4} is large enough.

The following property is very useful when we need to calculate the worst-case response time:

Property 5. For a given positive integer x , the minimum $t|t > 0$ such that $x(1 - \epsilon) + \lceil \frac{t}{2} \rceil + \lceil \frac{t}{4q} \rceil q + \lceil \frac{t}{8q} \rceil (2q - 1 + \epsilon) = t$ happens when t is $x \cdot 8q$.

Proof. This can be proved by simple arithmetics. \square

By using Property 5, it is not difficult to obtain the exact worst-case response time by using Lemma 1.

Lemma 8. Tasks $\tau_1, \tau_2, \dots, \tau_{m+3}$ in \mathbf{T}^{MILP} can meet their deadlines. The worst-case response time of task τ_n in \mathbf{T}^{MILP} is $16qm + (m - 1)(2q - 1)$.

Proof. The schedulability of tasks τ_1, τ_2, τ_3 comes by using the standard time demand analysis, and the schedulability of tasks $\tau_4, \tau_5, \dots, \tau_{m+3}$ follows from Property 5. The constructed task set \mathbf{T}^{MILP} has the following properties based on Lemma 1: a) We should always release the three highest priority tasks together with a computation segment of task τ_n and release their subsequent jobs periodically and as early as possible by respecting their minimum inter-arrival times until this computation segment of task τ_n finishes. b) If the response time of task τ_n is no more than $16qm^2 + (m - 1)(2q - 1)$, then each task τ_i for $i = 4, 5, \dots, m + 3$ only releases one job to interfere in a computation segment of task τ_n .

Suppose that there are ℓ_j tasks among $\tau_4, \tau_5, \dots, \tau_{m+3}$ which interfere in the j -th computation segment of task τ_n . We know that ℓ_j is an integer and $\ell_j \geq 0$ for $j = 1, 2, \dots, m$ and $\sum_{j=1}^m \ell_j = m$. Moreover, the response time of j -th computation segment of task τ_n is $(\ell_j + 1) \cdot 8q$ by Property 5. Therefore, we know that the worst-case response time of task τ_n is

$$\left(\sum_{j=1}^m (\ell_j + 1) \cdot 8q \right) + (m - 1)(2q - 1) = 16qm + (m - 1)(2q - 1).$$

Since $16qm + (m - 1)(2q - 1) < T_i$ for $i = 4, 5, \dots, n - 1$, we know that the above value is an upper bound by all the possible release patterns that satisfy Lemma 1. And, there is a concrete release/execution pattern which leads the response time of task τ_n exactly to this upper bound. Therefore, this is the exact worst-case response time of task τ_n in \mathbf{T}^{MILP} . \square

5.1 Excluding the Boundary Constraints by Eq. (2g) and Eq. (2h)

We first investigate whether the MILP without the boundary constraints presented by Eq. (2g) and Eq. (2h). We explore this specific condition under a special case, by further ignoring the interference of the tasks $\tau_4, \tau_5, \dots, \tau_{m+3}$. Then, the worst-case response time of the j -th computation segment of task τ_n (after the segment is released) can be

obtained by the following MILP.

$$\textbf{maximize: } R_j \quad (10a)$$

subject to:

$$R_j = 1 - \epsilon + N_{1,j} \cdot 1 + N_{2,j} \cdot q + N_{3,j} \cdot (2q - 1 + \epsilon), \quad (10b)$$

$$O_{1,j} \geq 0, O_{2,j} \geq 0, O_{3,j} \geq 0 \quad (10c)$$

$$0 \leq N_{i,j} \leq \left\lceil \frac{R_j - O_{i,j}}{T_i} \right\rceil, \quad \forall i = 1, \dots, 3 \quad (10d)$$

$$R_j > rel_{i,j} + \sum_{\ell=1}^3 \max \left\{ 0, \left\lfloor \frac{O_{\ell,j} + N_{\ell,j} T_{\ell} - rel_{i,j}}{T_{\ell}} \right\rfloor C_{\ell} \right\}, \quad \forall i = 1, \dots, 3 \quad (10e)$$

$$rel_{i,j} = O_{i,j} + (N_{i,j} - 1)T_i, \quad \forall i = 1, \dots, 3 \quad (10f)$$

$$N_{i,j} \text{ is an integer}, \quad \forall i = 1, \dots, 3 \quad (10g)$$

Lemma 9. *By the assumption that q is a positive integer $q \geq 1$ and $0 < \epsilon < 1/q$, the setting of $R_j = 8q^2 + 6q + 1 + q\epsilon$, $O_{1,j} = 0$, $O_{2,j} = \epsilon/4$, $O_{3,j} = \epsilon/2$, $N_{1,j} = 4q^2 + 3q + 1$, $N_{2,j} = 2q + 2$, and $N_{3,j} = q + 1$ is a feasible solution of the MILP in Eq. (10).*

Proof. The first condition in Eq. (10b) holds since

$$\begin{aligned} & 1 - \epsilon + 4q^2 + 3q + 1 + (2q + 2) \cdot q + (q + 1) \cdot (2q - 1 + \epsilon) \\ &= 1 - \epsilon + 4q^2 + 3q + 2q^2 + 2q + 2q^2 - q + q\epsilon + 2q - \epsilon + \epsilon \\ &= 8q^2 + 6q + 1 + q\epsilon. \end{aligned}$$

The conditions in Eqs. (10c), (10d), and (10g) clearly hold. In this case, the condition in Eq. (10f) sets $rel_{1,j} = O_{1,j} + (N_{1,j} - 1) \times 2 = 8q^2 + 6q$, $rel_{2,j} = O_{2,j} + (N_{2,j} - 1) \times 4q = 8q^2 + 4q + \epsilon/4$, and $rel_{3,j} = O_{3,j} + (N_{3,j} - 1) \times 8q = 8q^2 + \epsilon/2$. Now, we verify whether the condition in Eq. (10e) holds:

– When $i = 1$, we have

$$8q^2 + 6q + 1 < R_j.$$

– When $i = 2$, we have

$$\begin{aligned} & 8q^2 + 4q + \epsilon/4 + \max \left\{ 0, \left\lfloor \frac{8q^2 + 6q + 2 - (8q^2 + 4q + \epsilon/4)}{2} \right\rfloor \right\} \\ & + q + \max \left\{ 0, \left\lfloor \frac{8q^2 + 8q + \epsilon/2 - (8q^2 + 4q + \epsilon/4)}{8q} \right\rfloor (2q - 1 + \epsilon) \right\} \\ &= 8q^2 + 4q + \epsilon/4 + q + q + 0 = 8q^2 + 6q + \epsilon/4 < R_j \end{aligned}$$

– When $i = 3$, we have

$$\begin{aligned} & 8q^2 + \epsilon/2 + \max \left\{ 0, \left\lfloor \frac{8q^2 + 6q + 2 - (8q^2 + \epsilon/2)}{2} \right\rfloor \right\} \\ & + \max \left\{ 0, \left\lfloor \frac{8q^2 + 8q + \epsilon/4 - (8q^2 + \epsilon/2)}{4q} \right\rfloor q \right\} + 2q + 1 - \epsilon \\ &= 8q^2 + \epsilon/2 + 3q + q + 2q + 1 - \epsilon = 8q^2 + 6q + 1 - \epsilon/2 < R_j \end{aligned}$$

Therefore, we reach the conclusion. \square

Now, we can examine the MILP in Eq. (2), when excluding Eq. (2g) and Eq. (2h):

$$\textbf{maximize:} \quad S_n + \sum_{j=1}^m R_j \quad (11a)$$

subject to:

$$R_j = C_n^j + \sum_{i=1}^{n-1} N_{i,j} \times C_i, \quad \forall j = 1, \dots, m \quad (11b)$$

$$O_{i,j} \geq 0, \quad \forall i = 1, \dots, n-1, \forall j = 1, \dots, m \quad (11c)$$

$$O_{i,j+1} \geq O_{i,j} + N_{i,j} \times T_i - (R_j + S_n^j), \quad \forall i = 1, \dots, n-1, \forall j = 1, \dots, m-1 \quad (11d)$$

$$0 \leq N_{i,j} \leq \left\lceil \frac{R_j - O_{i,j}}{T_i} \right\rceil, \quad \forall i = 1, \dots, n-1, \forall j = 1, \dots, m \quad (11e)$$

$$R_j > rel_{i,j} + \sum_{\ell=1}^{n-1} \max \left\{ 0, \left\lfloor \frac{O_{\ell,j} + N_{\ell,j} T_\ell - rel_{i,j}}{T_\ell} \right\rfloor C_\ell \right\} \quad \forall i = 1, \dots, n-1, \forall j = 1, \dots, m \quad (11f)$$

$$rel_{i,j} = O_{i,j} + (N_{i,j} - 1)T_i \quad \forall i = 1, \dots, n-1, \forall j = 1, \dots, m \quad (11g)$$

$$N_{i,j} \text{ is an integer}, \quad \forall i = 1, \dots, n-1, \forall j = 1, \dots, m \quad (11h)$$

Lemma 10. Suppose that q is a positive integer $q \geq 1$ and $0 < \epsilon < 1/q$. For any $j = 1, 2, \dots, m$, the setting of $R_j = 8q^2 + 6q + 1 + q\epsilon$, $O_{1,j} = 0$, $O_{2,j} = \epsilon/4$, $O_{3,j} = \epsilon/2$, $N_{1,j} = 4q^2 + 3q + 1$, $N_{2,j} = 2q + 2$, and $N_{3,j} = q + 1$ is a feasible solution of the MILP in Eq. (11) for \mathbf{T}^{MILP} when $N_{i,j} = 0$, $O_{i,j} = 0$ for all $i = 4, 5, \dots, n-1$. Therefore, the optimal solution of Eq. (11) is at least $S_n + m(8q^2 + 6q + 1 + q\epsilon) = (m-1)(2q-1) + m(8q^2 + 6q + 1 + q\epsilon)$.

Proof. By Lemma 9, we only need to further verify whether the condition in Eq. (11d) holds when $i = 1, 2, 3$. By the definition $S_n^j = (2q-1)$, we know that $O_{i,j} + N_{i,j} \times T_i - (R_j + S_n^j) = O_{i,j} + N_{i,j} \times T_i - (8q^2 + 8q + q\epsilon) < 0$ for $i = 1, 2, 3$. Therefore, the condition in Eq. (11d) is by definition satisfied. \square

Now, we can reach the conclusion that MILP in Eq. (11) can be very far from the exact worst-case response time by the following theorem.

Theorem 3. The result of the MILP in Eq. (11) for task τ_n in task set \mathbf{T}^{MILP} divided by the exact worst-case response time of task τ_n is at least $\frac{m(8q^2+6q+1+q\epsilon)+(m-1)(2q-1)}{16qm+(m-1)(2q-1)}$. The ratio can become unbounded by the number of computation segments or the number of tasks when q is sufficiently large.

Proof. This follows directly from Lemmas 8 and 10. \square

Corollary 4. The result of the MILP in Eq. (2) by excluding the boundary constraints presented by Eq. (2g) and Eq. (2h) for task τ_n in task set \mathbf{T}^{MILP} divided by the exact worst-case response time of task τ_n is at least $\frac{m(8q^2+6q+1+q\epsilon)+(m-1)(2q-1)}{16qm+(m-1)(2q-1)}$.

Proof. This follows directly from Theorem 3. \square

5.2 Improvements by the Boundary Conditions Eq. (2g) and Eq. (2h)

We now discuss the complete MILP in Eq. (2). Calculating $UB_{ss,j}$ for task τ_n in \mathbf{T}^{MILP} is rather straightforward. This can be done by releasing all the jobs together with C_n^j . That is, $UB_{ss,j}$ is the minimum $t | t > 0$ such that $(m+1)(1-\epsilon) + \lceil \frac{t}{2} \rceil + \lceil \frac{t}{4q} \rceil q + \lceil \frac{t}{8q} \rceil (2q-1+\epsilon) = t$. By Property 5, we know that $UB_{ss,j}$ is $(m+1) \cdot 8q$.

Calculating UB_{ss} is tricky. However, for task τ_n in \mathbf{T}^{MILP} we can easily conclude that $UB_{ss} \leq m(m+1) \cdot 8q + S_n = 8qm(m+1) + (m-1)(2q-1)$. If we can get a very tight upper bound of UB_{ss} , then, there is no need of the MILP. Here is how UB_{ss} was proposed to be calculated by Nelissen et al. [21]:

Nelissen et al. [21]:⁸ *Constraints (2g) and (2h) reduce the research space of the problem by stating that the overall response time of τ_n and the response time of each of its execution regions, respectively, cannot be larger than known upper-bounds computed with simple methods such as the joint and split methods presented in [3].*

Lemma 11. *When q is set to m and $m \geq 2$, UB_{ss} derived from the joint and split methods presented in [3] is at least $8m^2(m+1) + (m-1)(2m-1)$ for \mathbf{T}^{MILP} .*

Proof. The joint and split methods presented in [3, Pages 131-141] are based on the following concept:

- A self-suspension interval of task τ_n can be converted to computation demand. (joint)
- A self-suspension interval of task τ_n can be treated as self-suspension, by considering their suffered worst-case interference independently. (split)

The following proof is only sketched since this can be easily proved by a simple observation. If we consider a self-suspension interval S_n^j as computation (i.e., the joint approach), then, the additional workload $(2q-1)$ (due to suspension as computation) increases the worst-case response time by $(2q-1)8q = (2m-1)8m = 16m^2 - 8m$. If we simply treat these two consecutive computation segments C_n^{j-1} and C_n^j by considering their suffered worst-case interference independently (i.e., the split approach), this treatment only increases the worst-case response time by at most $8m^2 + 2m - 1$. Therefore, the *joint* approach is always worse than the *split* approach, when $m \geq 2$. This can be formally proved by starting from $j = 1$ to convert any joint treatment to a split treatment in a stepwise manner.

Hence, $UB_{ss} = m(m+1) \cdot 8q + S_n = 8m^2(m+1) + (m-1)(2m-1)$ by splitting all the computation segments. \square

With the above discussions, we can reach the following lemma:

Lemma 12. *When q is set to m and $m \geq 2$, the objective function of the MILP in Eq. (2) for task τ_n in \mathbf{T}^{MILP} is at least $(m-1)(2m-1) + m(8m^2 + 6m + 1 + m\epsilon)$.*

Proof. Since $(m+1) \cdot 8q > 8m^2 + 6m + 1 + m\epsilon$ when q is set to m , we know that Eq. (2g) is satisfied by adopting the solution in Lemma 10. Similarly, since $8qm(m+1) + (m-1)(2q-1) > (m-1)(2m-1) + m(8m^2 + 6m + 1 + m\epsilon)$ when q is

⁸ The text is reorganized to use the proper references and notation in this paper.

set to m , we also know that Eq. (2h) is satisfied by adopting the solution in Lemma 10. Therefore, the solution in Lemma 10 is a feasible solution of the MILP in Eq. (2), in which we reach the conclusion of this lemma. \square

Theorem 4. *The result of the MILP in Eq. (2) (i.e., the MILP in [21]) for task τ_n in \mathbf{T}^{MILP} divided by the exact worst-case response time of task τ_n is at least $\frac{m(8m^2+6m+1+m\epsilon)+(m-1)(2m-1)}{16m^2+(m-1)(2m-1)} \geq \frac{4m+4}{9}$, when $m \geq 2$.*

Proof. This follows directly from Lemmas 8 and 12, and

$$\begin{aligned} & \frac{m(8m^2 + 6m + 1 + m\epsilon) + (m-1)(2m-1)}{16m^2 + (m-1)(2m-1)} = \frac{8m^3 + 8m^2 - 2m + 1 + m^2\epsilon}{18m^2 - 3m + 1} \\ &= \frac{4m+4}{9} + \frac{\frac{4m^2}{3} - \frac{10m}{9} + \frac{5}{9} + m^2\epsilon}{18m^2 - 3m + 1} > \frac{4m+4}{9}. \end{aligned}$$

\square

6 Conclusions and Discussions

This report shows that the schedulability analysis for fixed-priority preemptive scheduling even with only one segmented self-suspending task as the lowest-priority task is coNP-hard in the strong sense. Moreover, we also show that the upper bound on the worst-case response time by using a mixed-integer linear programming (MILP) formulation by Nelissen et al. [21] can be at least $\Omega(m)$ times the exact worst-case response time, where m is the number of computation segments of task τ_n .

Therefore, how to analyze the worst-case response time tightly remains as an open problem for self-suspending sporadic task systems even with one self-suspending sporadic task as the lowest-priority task under fixed-priority preemptive scheduling.

Acknowledgements. The author would like to thank Dr. Geoffrey Nelissen from CISTER, ISEP, Polytechnic Institute of Porto and Prof. Dr. Cong Liu from UT Dallas for their feedbacks on an earlier version of this report, which help the author improve the presentation flow and the clarity. This report is supported by DFG, as part of the Collaborative Research Center SFB876 (<http://sfb876.tu-dortmund.de/>).

References

1. N. C. Audsley and K. Bletsas. Fixed priority timing analysis of real-time systems with limited parallelism. In *16th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 231–238, 2004.
2. N. C. Audsley and K. Bletsas. Realistic analysis of limited parallel software / hardware implementations. In *10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 388–395, 2004.
3. K. Bletsas. *Worst-case and Best-case Timing Analysis for Real-time Embedded Systems with Limited Parallelism*. PhD thesis, Dept of Computer Science, University of York, UK, 2007.
4. K. Bletsas and N. C. Audsley. Extended analysis with reduced pessimism for systems with limited parallelism. In *11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 525–531, 2005.

5. J.-J. Chen and C. Liu. Fixed-relative-deadline scheduling of hard real-time tasks with self-suspensions. In *Proceedings of the IEEE 35th IEEE Real-Time Systems Symposium (RTSS)*, pages 149–160, 2014. **A typo in the schedulability test in Theorem 3 was identified on 13, May, 2015.** <http://ls12-www.cs.tu-dortmund.de/daes/media/documents/publications/downloads/2014-chen-FRD-erratum.pdf>.
6. J.-J. Chen, G. Nelissen, and W.-H. Huang. A unifying response time analysis framework for dynamic self-suspending tasks. In *28th Euromicro Conference on Real-Time Systems, ECRTS*, 2016.
7. J.-J. Chen, G. Nelissen, W.-H. Huang, M. Yang, B. Brandenburg, K. Bletsas, C. Liu, P. Richard, F. Ridouard, Neil, Audsley, R. Rajkumar, and D. de Niz. Many suspensions, many problems: A review of self-suspending tasks in real-time systems. Technical Report 854, Faculty of Informatik, TU Dortmund, 2016.
8. P. Ekberg and W. Yi. Uniprocessor feasibility of sporadic tasks with constrained deadlines is strongly coNP-Complete. In *27th Euromicro Conference on Real-Time Systems, ECRTS*, pages 281–286, 2015.
9. M. R. Garey and D. S. Johnson. *Computers and intractability: A guide to the theory of NP-completeness*. W. H. Freeman and Co., 1979.
10. W.-H. Huang and J.-J. Chen. Schedulability and priority assignment for multi-segment self-suspending real-time tasks under fixed-priority scheduling. Technical report, Dortmund, Germany, 2015.
11. W.-H. Huang and J.-J. Chen. Self-suspension real-time tasks under fixed-relative-deadline fixed-priority scheduling. In *Design, Automation, and Test in Europe (DATE)*, 2016.
12. W.-H. Huang, J.-J. Chen, H. Zhou, and C. Liu. PASS. In *Proceedings of the 52nd Annual Design Automation Conference on - DAC*. Association for Computing Machinery (ACM), 2015.
13. I. Kim, K. Choi, S. Park, D. Kim, and M. Hong. Real-time scheduling of tasks that contain the external blocking intervals. In *RTCSA*, pages 54–59, 1995.
14. J. Kim, B. Andersson, D. de Niz, J.-J. Chen, W.-H. Huang, and G. Nelissen. Segment-fixed priority scheduling for self-suspending real-time tasks. Technical Report CMU/SEI-2016-TR-002, CMU/SEI, 2016.
15. K. Lakshmanan and R. Rajkumar. Scheduling self-suspending real-time tasks with rate-monotonic priorities. In *Proceedings of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 3–12, 2010.
16. J. P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *RTSS*, pages 201–209, 1990.
17. J. P. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *IEEE Real-Time Systems Symposium*, pages 166–171, 1989.
18. C. Liu and J. Chen. Bursty-interference analysis techniques for analyzing complex real-time task models. In *Real-Time Systems Symposium (RTSS)*, pages 173–183, 2014.
19. C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the ACM*, 20(1):46–61, jan 1973.
20. L. Ming. Scheduling of the inter-dependent messages in real-time communication. In *Proc. of the First International Workshop on Real-Time Computing Systems and Applications*, 1994.
21. G. Nelissen, J. Fonseca, G. Raravi, and V. Nélis. Timing Analysis of Fixed Priority Self-Suspending Sporadic Tasks. In *Euromicro Conference on Real-Time Systems (ECRTS)*, 2015.
22. J. C. Palencia and M. G. Harbour. Schedulability analysis for tasks with static and dynamic offsets. In *Proceedings of the 19th IEEE Real-Time Systems Symposium (RTSS)*, pages 26–37, 1998.
23. F. Ridouard, P. Richard, and F. Cottet. Negative results for scheduling independent hard real-time tasks with self-suspensions. In *RTSS*, pages 47–56, 2004.